

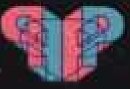
5

Avec la cassette de vidéo-jeux et programmes pour ZX SPECTRUM et VIC 20

PSEUDOCODE:
- NOMBRES ALEATOIRES
- LA DISTRIBUTION DE CARTES

COURS DE BASIC

LE LANGAGE MACHINE



Promopublications

M 6111-5 - 85 F Belgique: 680 FB Suisse: 28 FS Canada: 14$ C

SOFTHEQUE N. 5

MENSUEL - JUILLET 1985

Directeur: Franco Bozzesi

3 PSEUDOCODE: NOMBRES ALEATOIRES - LA DISTRIBUTION DE CARTES

10 COMMENT TRADUIRE UN NOMBRE DECIMAL EN BASE 2 OU 16

14 LE BASIC DU SPECTRUM LES FONCTIONS PRINCIPALES

21 LE LANGAGE MACHINE

30 INSTRUCTIONS POUR LA CASSETTE DE SOFTHEQUE N. 5

Ont collaboré:
MICHELLE BLEIN
GEORGES RIEBEN
MAX CELLINI
ROBERTO TREPPIEDI
DIDIER DUCHESNE
ALDO CAMPANOZZI
GEORGETTE LHOPITAL
ALEX VALLONE
HELENE RACCAH
ALBERTO BARBATI
YOLANDE TERISSE
ANTONIO LUCARELLA
CATHERINE JUERY
DANIELE RIEFOLI
JEAN CAPOBIANCO

SOFTHEQUE est une création de PROMOPUBLICATIONS, Sarl au capital de 20.000 F. 312179195 B.R.C. Paris.
Rédaction, administration, vente, publicité, siège social: 34, Champs-Elysées, 75008 Paris. Tel. (1) 5634850
Distribué en France par: N.M.P.P.
Imprimerie: ALIGRAF Milan - Italie
Directeur de la publication: Franco Bozzesi
Numéro de commission paritaire: en cours.
Depôt legal n. 50579 du 8 Juin 1984.

NOMBRES ALEATOIRES LA DISTRIBUTION DE CARTES

Nous voici arrivés au cinquième article de la série. Désormais vous possédez un bagage de connaissances suffisant pour vous permettre d'affronter un problème complexe et en définir la solution.

Le problème que nous présentons ce mois-ci est la DISTRIBUTION D'UN JEU DE 52 CARTES à 4 joueurs. Malgré l'apparente simplicité du problème, vous verrez que le programme est plutôt riche en trouvailles! Considérez également que le temps d'exécution sur un O.P. est de plus de 40 secondes: cela signifie que ce problème est vraiment puissant du point de vue calcul.

En outre sous ce problème se cache un thème important: le HASARD des évènements impliqués.

C'est justement ce HASARD qui nous guidera dans tout cet article. Comme d'habitude, nous ne vous proposons pas seulement des solutions, mais nous tenons à vous suggérer des idées d'étude.

Enfin le programme BASIC qui accompagne cet article, peut être utilisé comme le point de départ d'une plus ambitieuse réalisation pour n'importe quel jeu de cartes par un joueur "automatique".

LE HASARD

Un programme est un ensemble d'instructions claires et non-ambigues. Le pseudocode enseigne que la logique du programme est bien représentée par les figures fondamentales et que les prédicats de contrôle fonctionnent comme d'impitoyables cerbères.

Comment pouvons-nous alors représenter le HASARD propre à certains évènement naturels à l'intèrieur d'un programme?

Rappelons que le HASARD d'un évènement est la propriété de cet évènement à être incertain: c'est-à-dire de pouvoir se manifester d'une certaine manière ou d'une autre manière complètement différente. Ou bien même de ne pas se manifester du tout.

Un exemple typique est le choix d'une carte dans un jeu de 52 cartes. Dans cet exemple nous appelons "EVENEMENT" le fait qu'une certaine carte soit extraite du jeu. Dans la DISTRIBUTION de 52 cartes à 4 joueurs 52 de ces évènements

peuvent se passer.
Essayons de voir comment nous pourrions programmer une simulation de distribution de cartes à travers un programme en pseudocode.
Si vous y pensez 5 minutes vous vous rendrez compte qu'il est IMPOSSIBLE d'écrire un programme qui réalise un choix VRAIMENT fortuit parmi 52 cartes en utilisant uniquement des instructions DETERMINISTES, comme celles étudiées jusqu'à présent, sans que nous puissions savoir EXACTEMENT quel choix sera fait par notre programme, et cela détruit le hasard de l'évènement.
Une approche de solution pourrait être la suivante:

DEBUT. Choix au hasard entre 52 cartes.
"Obtenez du clavier le nombre N"
"Choisissez la Nième carte du jeu"
FIN

Avec l'instruction INPUT du nombre N nous laissons le choix de la carte à la personne assise devant l'O.P.
Ainsi celui qui émet le nombre assume la responsabilité de choisir au hasard. ON RENONCE donc A PROGRAMMER le hasard, puisque l'on délègue ce choix à quelqu'un.
Cependant nous ne serons jamais satisfaits, car nous saurons toujours quelle carte il choisira pour chaque N (l'Nième est toujours la même).
Comment sortir de ce cercle infernal?
La solution existe et consiste à se mettre dans une perspective différente. En réalité le HASARD existe parce que nous ne sommes pas capables de prévoir avec sûreté si un certain évènement se manifestera ou non. En fait nous avons l'impression du hasard lorsque nous ne savons pas prévoir le résultat qui suivra une action.
En effet, en ce qui concerne le choix d'une carte dans un jeu de 52, si nous connaissions exactement la disposition de toutes les cartes après les avoir battues, notre choix ne serait plus FORTUIT mais conscient et voulu, de même en ce qui concerne le programme où nous choisissions "au hasard" un nombre, nous connaissions en fait la carte sélectionnée.
Voici donc le chemin à suivre pour résoudre le problème: écrire un programme qui engendre un nombre dont nous ne pouvons absolument pas prévoir ce qu'il sera, et ce nombre sera utilisé pour choisir une carte. Je crois que beaucoup d'entre vous penseront que je me moque d'eux: "mais comment, diront-ils, le hasard de l'évènement appartient à l'évènement et ne dépend pas du fait que je sache ou non que celui-ci arrive: l'incertitude est dans l'évènement et non en moi-même!!"
Ceux-ci peuvent s'énorgueillir de cette opinion philosophique, car ils sont en bonne compagnie: Einstein lui-même pensait



ainsi mais seulement... "comme ligne de principe".
Dans la pratique il est démontré mathématiquement que cette position et celle décrite plus haut sont équivalentes dans leurs effets: que l'incertitude se trouve dans l'évènement ou en qui observe l'évènement, il n'y a pas de différence pratique.

Dans l'une ou l'autre des deux attitudes "philosophiques" les choses restent semblables. Naturellement nous choisissons la seconde attitude car c'est la seule programmable.
L'ensemble des programmeurs appelle ALEATOIRE n'importe quel programme qui engendre des nombres sans qu'on puisse prévoir (avec certitude) quel nombre successif le programme engendrera.

"Mais comment" diront certains "même pas celui qui a écrit le programme ne connaît le nombre que le programme extraira?" La réponse est NON, même pas lui. En analysant un de ces programmes extracteurs de nombres ALEATOIRES (en anglais RANDOM) on voit qu'il existe cependant une certaine loi sous-jacente. Seulement celle-ci est tellement bizarre et complexe qu'aucun être humain ne peut avoir une perception "instinctive" d'une loi sous-jacente. Le programmeur qui voit le programme extraire un nombre, le voit agir absolument au HASARD.

LES NOMBRES ALEATOIRES (RANDOM) EN BASIC

En BASIC avec une seule instruction (qui rappelle un programme déjà écrit) on engendre un nombre compris entre 0 et 1. Par exemple en écrivant:

```
10 A = RND
20 PRINT A
```

et en lançant le programme plusieurs fois on obtient des nombres pas exemple comme 0.538 ou bien 0.997, bref n'importe quel nombre entre 0 et 1 inclus.
Pour obtenir un nombre dans un intervalle supposons entre 0 et 50, il suffit de multiplier le nombre produit par RND par 50.
Pour obtenir un nombre entre 1 et 52 nous devons alors écrire:
A = (RND*52) + 1.
Si nous voulons un nombre ENTIER (c'est-à-dire sans virgule) entre 1 et 52, nous écrirons:
A = INT (RND*52) + 1.
Notre problème du choix de la carte peut être résolu de la manière suivante:

DEBUT. Choix d'une carte sur 52
"Attribue à N un nombre random compris entre 1 et 52"
"Choisissez la carte Nième"
FIN

En lançant ce programme, le programmeur ne pourra pas prévoir la carte qu'extraira le programme, exactement comme s'il demandait à un ami de choisir un nombre au hasard.

LA DISTRIBUTION DES CARTES

Le problème de distribuer 52 cartes à 4 joueurs n'est pas une simple répétition de 52 choix d'une carte.
Il y a une grande différence dont il faut tenir compte: après chaque extraction il faut trouver le moyen de "La supprimer du jeu": celle-ci ne devant plus être extractible.
Ensuite remarquons que nous voulons un système pour représenter les 52 cartes du jeu afin de ne pas être obligé de faire 52 attributions peu pratiques.
Nous pensons représenter une carte au moyen de deux variables indicées: C(1) qui représente la COULEUR de la 1ère carte, et V(1) qui en représente la VALEUR.
Pour utiliser des variables numériques (de façon à gérer les valeurs de manière numérique) utilisons les nombres 1, 2, 3 et 4 pour représenter respectivement Pique, Coeur, Carreau et Trèfle.
De la même façon les nombres 13, 12 et 11 représentent respectivement Roi, Dame et As.
A ce point le chargement (la préparation) du jeu ordonné peut se faire ainsi:

```
DEBUT. Chargement du jeu
  REPETEZ
  POUR I DE 1 A 4
   REPETEZ
   POUR J DE 1 A 13
    K = (I—1)*13+J
    C (K) = I
    V (K) = J
   FIN
  FIN
FIN.
```

Ce programme introduit dans C (K) et V (K) toutes les cartes ordonnées de Pique à Trèfle. La gestion de K est amusante et je laisse le lecteur en comprendre la logique (un conseil: de cette façon K assume les valeurs de 1 à 52 en série).
Après avoir créé le jeu pensons à la distribution:

```
DEBUT. Distribution
  REPETEZ
  POUR I DE 1 A 13
   REPETEZ
   POUR J DE 1 A 4
   "Choisissez une carte
   parmi celles qui restent"
   "Donnez la carte au joueur J"
   "Eliminez du jeu la carte
   extraite"
   FIN_REPETEZ
  FIN_REPETEZ
FIN.
```

(Observez le choix du joueur Jième, lorsque J oscille entre 1 et 4). Pour chaque joueur nous devons penser comment mémoriser celui qui a reçu la carte extraite.
Pensons à une variable à double indice, le premier indique le joueur, le second le nombre progressif de la série de cartes qui lui sont distribuées. Cependant pour chaque carte nous voulons connaître la valeur et la couleur. C'est pourquoi nous aurons une paire de variables à deux indices: appelons-les Z (H,L) et G (H,L). Par exemple si



Z(4,10) = 2 et G(4,10) = 1 alors le joueur 4 a comme 10ième carte en main un as de carreau. Compris? Un autre exemple: Z(3,2) = 1 et G(3,2) = 11 cela signifie que la 2ième carte du 3ième joueur est un valet de pique.
Voici comment on reécrit notre programme, avec ces hypothèses:

```
DEBUT. Distribution
  REPETEZ
  POUR I DE 1 A 13
   REPETEZ
   POUR J DE 1 A 4
    "Choisissez une carte au
    hasard: la carte avec
    indice X"
    Z (J,I) = C (X)
    G (J,I) = V (X)
    "Supprimez la carte du jeu"
   FIN_REPETEZ
  FIN_REPETEZ
FIN.
```

Observez le jeu de J et de I, qui comptent respectivement les joueurs et les tours.
Nous sommes arrivés au problème crucial: comment choisir dans le jeu une carte parmi 52 au premier tour, une parmi 51 au second...jusqu'à 1 parmi 2 à l'avant-dernier tour?
Il faut introduire une variable P à la place de 52 dans l'instruction décrite au paragraphe précédent:
X = INT(RND*(P)) + 1
et ensuite faire décroîte P de 52 à 1, enlevant 1 à chaque tour.
Le dernier problème à résoudre est comment supprimer du jeu une carte déjà choisie? Voici une idée. Une fois choisie la carte X et attribuées les valeurs au joueur J, tournée 1, réordinons les vecteurs C(N) et V(N). Déplaçons tout le jeu en haut d'une carte d'une place depuis la carte X + 1. De cette façon, au choix suivant, entre 1 et (P-TOUR) nous tomberons toujours sur une carte non encore choisie. Encore mieux, si nous mettons 0 tant en Couleur qu'en Valeur dans la dernière carte du jeu. Ainsi peu à peu les vecteurs se remplissent de zéros à la fin, indiquant les cartes éliminées.
Voici le pseudocode:

```
DEBUT_Supprimez la carte X
du jeu.
  REPETEZ
  POUR N DE 1 A 52
   V(N) = V(N + 1)
   C(N) = C
   (N + 1)
  FIN_REPETEZ
FIN
```

(Afin d'utiliser ce truc avec élégance, il convient de mettre une 53ième carte fictive dans le jeu dont la valeur et la couleur valent 0.
A présent les puzzles de notre programme sont tous placés.
Il suffit de créer le programme d'édition des cartes des joueurs à la fin de la distribution des cartes.
Pour cela nous utilisons un vecteur de décodification: pour chaque nombre de V nous attribuons un caractère ASCII qui en représentent la carte. La même chose pour la couleur.

Appelons le premier vecteur V$, et le second R$.

Alors nous pouvons également penser à regrouper les cartes par Couleur in output. Nous obtenons:

```
DEBUT Impression
REPETEZ
    POUR I DE 1 A 4
    Impression "Joueur" I
     Impression Couleur: R$(J)
    POUR K DE 1 A 13
     SI Z (I,K) = J
     ALORS Imprimez V$
     (G(I,K))
    FIN_REPETEZ
   FIN_REPETEZ
FIN
```

Terminé, le programme est projeté. Celui qui le désire peut s'amuser à le réaliser tout seul. Pour celui qui en a..."marre" le programme BASIC est déjà prêt figure 1

Mettez-le en mémoire et faites-le tourner. Le résultat est semblable à celui de la figure 2, mais lisez attentivement le sous-titre.

Apparemment l'instruction 131 n'a pas beaucoup de sens. Mais essayez de l'enlever... Vous aurez l'émotion d'attendre avec "suspense" que le programme termine ses 40 secondes (une éternité) d'élaboration.

Au revoir.

```
  10 DIM C(53): DIM V(53): DIM G
(4,13): DIM Z(4,13): DIM R$(4):
DIM V$(13)
  15 LET R$(1)="P": LET R$(2)="C
": LET R$(3)="Q": LET R$(4)="F"
  16 LET V$(1)="1": LET V$(2)="2
": LET V$(3)="3": LET V$(4)="4"
  17 LET V$(5)="5": LET V$(6)="6
": LET V$(7)="7": LET V$(8)="8"
  18 LET V$(9)="9": LET V$(10)="
10": LET V$(11)="J": LET V$(12)=
"Q": LET V$(13)="K"
  20 REM DISTRIBUTION
  30 GO SUB 2000: REM RAPPEL CHA
RGEMENT-CARTES
  40 LET P=52: REM NOMBRE DE CAR
TES RESTANTES DANS LE JEU
  50 FOR I=1 TO 13
  60 FOR J=1 TO 4
  80 LET X=INT (RND*(P))+1
  90 LET Z(J,I)=C(X)
 100 LET G(J,I)=V(X)
 110 GO SUB 3000: REM RAPPEL SUP
PRESSION CARTES
 120 LET P=P-1
 130 NEXT J
 131 PRINT "TOUR NUMERO ...";I
 140 NEXT I
 150 REM IMPRIME LES CARTES DIST
RIBUEES
 160 FOR I=1 TO 4
 170 PRINT "JOUEUR NUMERO ";I
```




```
 180 FOR J=1 TO 4
 190 PRINT R$(J);"  :";
 200 FOR K=1 TO 13
 210 IF Z(I,K)=J THEN PRINT V$(G
(I,K));"  ";
 220 NEXT K
 225 PRINT
 230 NEXT J
 240 NEXT I
 250 STOP
2000 REM CHARGE LE JEU DE CARTES
2010 FOR I=1 TO 4
2020 FOR J=1 TO 13
2030 LET K=(I-1)*13+J
2040 LET C(K)=I
2050 LET V(K)=J
2060 NEXT J
2070 NEXT I
2075 LET C(53)=0:  LET V(53)=0
2080 RETURN
3000 REM SUPPRIME UNE CARTE DE J
EU
3010 FOR N=X TO 52
3030 LET V(N)=V(N+1)
3035 LET C(N)=C(N+1)
3040 NEXT N
3050 RETURN
```

```
JOUEUR NUMERO 1
P    :9   1
C    :4   5   8
Q    :7   K   1   9
F    :4   8   10  K
JOUEUR NUMERO 2
P    :10  7   6   J   K
C    :1   10  7   K
Q    :J
F    :5   3   1
JOUEUR NUMERO 3
P    :2   5   8   Q
C    :3   J
Q    :8   10  6   4   Q
F    :Q   2
JOUEUR NUMERO 4
P    :4   3
C    :Q   2   6   9
Q    :3   2   5
F    :7   9   J   6
BREAK IN 250
OK
```

Exemple d'une distribution de cartes du programme DISTR.. Celui écrira son programme personnel obtiendra difficilement cette distribution: il y a la même probabilité d'obtenir les mêmes cartes lors de 2 distributions successives!

"COMMENT TRADUIRE UN NOMBRE DECIMAL EN BASE 2 OU 16"

Les nombres décimaux nous sont familiers. Ils se divisent en deux catégories: les nombres entiers (sans virgule) et ceux avec la virgule.
Maintenant voyons comment convertir un nombre décimal (avec ou sans virgule!) en représentation en n'importe quelle autre base (comprise entre 2 et 16).
Le programme BASIC habituel vous aidera à approfondir les concepts exposés.

REPRESENTATION DES NOMBRES

Dans le numéro précédent nous avons parlé des "systèmes de numération" décimal, binaire, octal et hexadécimal.
Après vous être exercé avec le programme Basic dans les différentes représentations, vous êtes maintenant capables de passer d'un système de numération à un autre et vous avez peut être aussi appris à utiliser à meilleur escient l'habituelle représentation décimale".
L'information à l'intèrieur de l'élaborateur est représentée en binaire, l'extèrieur au contraire, en décimal. Le passage de l'information entre l'intèrieur et l'extèrieur d'un élaborateur est assuré par l'unité d'entrée et sortie, qui prévoit à convertir l'information de représentation binaire à décimale et vice-versa. Par la suite nous parlerons de ces systèmes, connus sous le nom CODES.
Lorsque vous deviendrez des programmeurs chevronnés il vous arrivera d'avoir besoin de lire certaines informations directement de la représentation intèrieure afin de corriger des erreurs ou tout simplement par curiosité.
Voyons maintenant comment il est possible de passer de la représentation décimale à la représentation binaire (vous êtes déjà capables de faire le contraire...).
Afin de convertir un nombre entier décimal en un nombre binaire correspondant (octal ou hexadécimal) on utilise la méthode suivante:
Méthode de la division: On



divise le nombre entier décimal par la "base" du système de numération que nous voulons obtenir (2, 8 ou 16): on met le reste de la division comme chiffre binaire (octal ou hexadécimal) le moins significatif (de droite à gauche) du nombre binaire correspondant que nous sommes en train de construire. Ensuite on divise encore le résultat de la première division par la "base" et on répète le procédé, obtenant au fur et à mesure les autres chiffres.
Le processus s'arrête lorsque le résultat d'une division devient égal à 0.
Nous voulons, par exemple, représenter le nombre 183, en binaire, octal et hexadécimal: nous reportons à la suite les "résultats partiels" des divisions et le reste.

BASE 2

VALEUR INITIALE ET RESULTATS PARTIELS	RESTE
183:2	
91	1
45	1
22	1
11	0
5	1
2	1
1	0
0	1

$183_{10} = (10110111)_2$

BASE 8

	RESTE
183:8	
22	7
2	6
0	2

$183_{10} = (267)_8$

BASE 16

183:16	
11	7
0	B

$183_{10} = (B7)_{16}$

Jusqu'ici nous avons vu comment transformer un nombre entier décimal en un nombre binaire (octal et hexadécimal): mais comme vous le savez les nombres entiers ne traitent pas à fond l'ensemble des nombres que nous utilisons habituellement: il y a aussi les nombres fractions (ex. 10/5). Ensuite nous analyserons comment transformer ces nombres; en ce qui concerne la partie entière du nombre (ex. 10.5, partie entière = 10, partie décimale = 5) elle est transformée selon la méthode de la division illustrée précédemment tandis que la partie décimale est convertie avec la "méthode de la multiplication".
Cette méthode peut être schématisée ainsi.
On multiplie la partie fractionnaire du nombre par 2 (8 ou 16) on prend la partie entière du produit obtenu et on le met comme chiffre binaire (octal ou hexadécimal) le plus significatif (attention dans ce cas de gauche à droite) du nombre binaire correspondant: on procède en mode interactif en multipliant la partie décimale du produit obtenu, et en s'arrêtant

lorsque l'on atteint la précision désirée.
Observez que cette méthode ne s'arrête pas toujours à zéro comme la précédente, mais c'est à nous de décider d'arrêter la représentation du nombre au troisième, quatrième... chiffre que nous considérons comme significatif.
Considérons le nombre 0.250 dont nous voulons obtenir la représentation binaire; ce nombre a 3 chiffres dans la partie fractionnaire, décidons alors d'arrêter aussi la méthode de la multiplication au troisième chiffre binaire (ou avant si nous trouvons le zéro).

```
0250        2  =  0.5  0
0.5               1.0  1
0.0               0    0
(0.250)10 (0.010)2
```

De même dans ce numéro le programme basic vous aidera à mieux comprendre ce qui vous a été expliqué dans les pages précédentes, vous permettant ainsi de vérifier à chaque instant si le nombre binaire que vous avez obtenu est réellement la conversion du nombre décimal de départ; exercez-vous et alors vous serez prêts pour affronter le prochain argument. Arithmétique binaire.

LE PROGRAMME BASIC

Voyons la réalisation en BASIC de la conversion de DECIMAL en une autre base pour des nombres entiers.
Comme limites, nous ne convertissons pas de nombres qui comptent plus de 8 chiffres dans la nouvelle base.
Le programme est la réalisation BASIC de la méthode illustrée dans les pages précédentes. Cette méthode se base sur des divisions successives et déterminations des restes. La série des restes mis les uns à la suite des autres (dans l'ordre contraire où ils sont engendrés), est la représentation du nombre dans la nouvelle base. On se sert donc d'une variable R(1) pour contenir les chiffres du nombre dans la nouvelle base.
La première fois le nombre à diviser (dividende) est DEC, c'est-à-dire le nombre introduit (instruction 307).
Ensuite, le quotient de la division précédente devient le nouveau dividende. Rappelons-nous que le quotient est l'entier plus grand contenu dans le résultat de la division, et que le diviseur est la base choisie (instruction 20).
En substance, le quotient se trouve avec l'instruction 60 et est mémorisé dans A. (Observez que la première fois Q est égal à DEC). Le reste de la division est ensuite calculé par l'instruction 70. L'instruction 75 met le nouveau dividende (Q) à la valeur du quotient de la division (A).



La série d'instructions 60-75 sont répétées (en vertu de 60, 80 et 90) jusqu'à ce que Q devienne O.
Le résultat est présenté en émettant les chiffres trouvés en parcourant R(1) en arrière.
Une dernière explication: puisque R(1) contient des nombres et non pas des chiffres de systèmes en base supèrieure à 10, il faut au contraire émettre pour ces chiffres les caractères A,B,C etc. comme nous l'avons déjà expliqué dans l'article précédent.
Ceci nous induit à considérer chaque chiffre comme représenté par un caractère et à construire le vecteur de conversion $$(1).
Observez que, puisque R(1) peut valoir 0 et 0 ne peut être un indice (instruction 110), il faut ajouter 1 à R(1) pour obtenir le chiffre correspondant dans le vecteur de correction.

```
10 DIM R(20),S$(16)
13 S$(1)="0":S$(2)="1":S$(3)="2":S$(4)="3":S$(5)="4":S$(6)="5"
15 S$(7)="6":S$(8)="7":S$(9)="8"
16 S$(10)="9":S$(11)="A":S$(12)="B":S$(13)="C":S$(14)="D":S$(15)="E":S$(16)="F"
20 PRINT"INTRODUISEZ BASE":PRINT:INPUTBASE
25 PRINT
30 PRINT"INTRODUISEZ NOMBRE    ENTIER: ":PRINT:INPUT DEC
40 Q=DEC:I=1:R(1)=0
50 IF Q=0 GOTO 99
60 A=INT(Q/BASE)
70 R(I)=(Q-(BASE*A))
75 Q=A
80 I=I+1
90 GOTO 50
99 REM    FIN REPETEZ
100 FOR K=I TO 1 STEP -1
110 PRINT S$(R(K)+1);
120 NEXT K
READY.


INTRODUISEZ BASE
 2

INTRODUISEZ NOMBRE    ENTIER: 255
011111111
READY.


INTRODUISEZ BASE
 16

INTRODUISEZ NOMBRE    ENTIER: 16738
04162
READY.
```

LES FONCTIONS PRINCIPALES

La dernière fois nous avons étudié les instructions fondamentales du BASIC et nous avons également parlé des variables indicées.
Cette fois-ci, nous nous occuperons de plusieurs instructions d'utilisation moins courantes et de toutes les fonctions dont dispose le BASIC du SPECTRUM.
Vous vous en souvenez: Afin de permettre à l'ordinateur de recevoir des données pendant l'exécution d'un programme nous avons toujours utilisé l'instruction INPUT, mais ce n'est pas la seule qui permette d'interagir avec l'opérateur. En effet il existe aussi l'instruction INKEY$, qui, à la différence de INPUT, ne peut recevoir un ensemble de données suivies par la pression de ENTER. Elle "lit" la touche qui a été pressée et en attribue éventuellement la valeur à la variable spécifiée. Par exemple, l'instruction:

```
LET A$ = INKEY$
```

reconnaît la touche qui a été pressée et en attribue le caractère correspondant à la variable A$. Si aucune touche n'était pressée au moment de l'exécution de cette instruction on attribuera à la variable A$ une valeur nulle.
Pour demeurer dans le thème des variables, domaine par ailleurs assez vaste, citons une autre manière d'attribuer des valeurs à une variable, mais cette fois-ci à l'intèrieur du programme où les données intéressées doivent être écrites pendant l'élaboration du programme. L'instruction concernée est READ qui en anglais signifie justement LISEZ.
Prenons le cas d'un programme où nous devons employer de grandes quantités de données toujours égales et qui ne doivent donc pas être fréquemment mises à jour. Nous pouvons mettre ces données dans un programme grâce à l'instruction DATA suivie justement des données qui nous intéressent.
Celles-ci peuvent être numériques ou alphanumériques et séparées entre elles par une virgule. Ces données seront ensuite lues en séquence par l'instruction READ suivie du nom de la variable à laquelle on désire attribuer la donnée. Faites attention à ce que le type de variable que vous utilisez corresponde effectivement au type de donnée qu'elle doit lire. Rappelez-vous aussi que les chaînes



doivent être mises entre guillemets.
Etudions maintenant un exemple simple.

```
10 FOR A=1 TO 7
20 READ S$
30 PRINT S$
40 NEXT A
50 DATA "LUNDI", "MARDI"
60 DATA "MERCREDI",
   "JEUDI"
70 DATA "VENDREDI",
   "SAMEDI"
80 DATA "DIMANCHE"
```

Comme vous l'aurez probablement compris, la ligne 10 commence un cycle qui va de 1 à 7 à l'intèrieur duquel se trouve une instruction READ qui lit les valeurs contenues dans les lignes de DATA. Elle les attribue à la variable chaîne S$ laquelle est imprimée immédiatement après. Comme vous constaterez, dans les lignes de DATA qui suivent le programme principal nous avons mis le nom des jours de la semaine (exemple typique de donnée constante). Pour une meilleure clarté nous n'avons introduit que deux jours pour chaque instruction DATA. Il est cependant possible d'en mettre davantage mais nous le déconseillons car, plus la ligne est longue et plus l'ordinateur met de temps pour déplacer le curseur. De toute façon peu importe que les données se trouvent sur des lignes différentes, car l'instruction READ, où qu'elle soit placée, pointe toujours la donnée successive à celle qui vient d'être lue.
Cependant si nous devions lire plusieurs fois les mêmes données nous ne pourrions le faire qu'a l'aide d'une instruction spéciale appelée RESTORE, qui reporte les pointeurs à la première donnée de la première instruction DATA. Ce qui permet ainsi de relire l'ensemble autant de fois que nous le voulons. Prenons un exemple facile:

```
10 PRINT"NOMBRES
   PREMIERS"
20 FOR A=1 TO 5
30 READ N
40 PRINT N
50 NEXT A
60 LET P$=INKEY$
65 IF P$ = "  " THEN GOTO
   60
70 RESTORE:CL$
75 GOTO 10
90 DATA 1,3,5,7,11
```

Ce bref programme imprime les cinq premiers nombres premiers qui sont mémorisés dans la ligne 90 par un cycle semblable au précédent. Après quoi les lignes 60 et 65 attendent la pression d'une touche et ensuite les lignes 70 et 75 qui reportent les pointeurs de données et effacent l'écran sont exécutées retournant ensuite à la ligne 10.
Nous terminerons en vous faisant observer que les lignes de DATA ne doivent pas obligatoirement se trouver à la fin du programme, mais puisqu'elles ne demandent aucune opéra-

tion il est inutile de les mettre au beau milieu du programme. Cela provoquerait uniquement un rallentissement dans les temps d'exécution. On préférera les placer à la fin pour des raisons esthétiques et pour une lecture plus facile du programme de la part d'une autre personne.

A présent que vous avez appris à créer des programmes BASIC assez difficiles, il ne vous reste qu'à apprendre à "sauvegarder" et à "charger" les programmes sur une cassette normale. Si vous avez un programme en mémoire, vous devez seulement introduire dans le magnétophone relié à votre ordinateur une cassette vierge (ou du moins dont vous ne vous servez pas) et écrire ensuite la commande SAVE "nom", où "nom" peut être n'importe quelle chaîne composée au maximum de 10 caractères qui représentent le nom du programme.

Tapez maintenant sur la touche ENTER et vous verrez apparaître l'inscription:

START TAPE THEN PRESS A KEY

ce qui signifie

FAITES MARCHER VOTRE MAGNETOPHONE ET APPUYEZ SUR UNE TOUCHE.

Appuyez simultanément RECORD et PLAY sur le magnétophone et ensuite pressez une touche, vous verrez des lignes se déplacer sur les bords de l'écran, signe que l'ordinateur est en train de "sauvegarder" sur bande le programme BASIC. Cette opération terminée le message 0:OK apparaîtra. A présent il vaudrait mieux vérifier si l'enregistrement des données est correct afin d'éviter des surprises désagréables; on peut le faire par la commande VERIFY "nom", laquelle compare ce qui est enregistré sur bande avec les données présentes en mémoire et, si elle trouve des erreurs elle vous le signalera.

Dans ce cas répéter l'opération de sauvetage, en cas contraire le message OK apparaîtra. Maintenant votre programme, ou celui de quelqu'un d'autre, est mémorisé définitivement sur la cassette et vous pourrez le recharger quand vous voudrez au moyen de la commande LOAD "nom".

A l'exécution de cette commande le contour de l'écran changera de couleur et, après avoir mis en route le magnétophone, clignotera jusqu'à ce qu'il trouve un signal (représenté par des raies colorées); alors il imprimera sur vidéo le nom du programme trouvé et si celui-ci correspond au nom que vous avez spécifié dans la commande LOAD le programme sera chargé. Faites très attention pendant le sauvetage des données, car le petit câble doit être disjoint d'un côté au moins; en cas contraire le programme ne sera pas sauvegardé.

Si vous désirez qu'un pro-



gramme à peine chargé parte seul, vous devrez spécifier le numéro de la ligne à partir de laquelle l'exécution devra commencer directement avec la commande SAVE. La syntaxe pour définir la ligne d'auto-départ dans une commande SAVE est la suivante

SAVE "nom" LINE N

où N est le numéro de la ligne à partir de laquelle commence l'exécution du programme.
Avec le Spectrum il est également possible de sauvegarder des matrices de variables numériques ou alphanumériques ou encore des parties de mémoire sous forme d'octets.
Si dans un programme BASIC nous décidons d'utiliser une variable indicée pour classer quelque chose (par exemple des adresses d'amis), nous pourrons après sauvegarder toute la matrice et la recharger ensuite, la modifier et la sauvegarder à nouveau, à notre plaisir. L'instruction pour sauvegarder une matrice est:

SAVE "nom" DATA X ( )

où X est le nom de la matrice (suivie de $ s'il s'agit de chaînes). Les procédés de vérification et de chargement sont les suivants:

VERIFY "nom" DATA X( )
LOAD "nom" DATA X( )

Rappelez-vous toutefois de dimensionner la matrice intéressée avant de la charger.
Nous terminons en donnant quelques indications sur les fonctions de corrections sur le Spectrum. Si vous avez un programme BASIC en mémoire, vous pourrez le voir grâce à la commande LIST laquelle donne le LISTING (c'est ainsi que s'appelle l'ensemble des instructions et les numéros des lignes). Chaque fois que l'écran est rempli l'ordinateur vous posera la question SCROLL? si vous ne voulez pas que l'ordinateur poursuive l'examen du listing, appuyez sur la touche SPACE ou N, ou n'importe quelle autre touche. Pour rappeler une ligne désirée écrivez:

LIST N

où N est la ligne intéressée; appuyez sur SPACE si l'ordinateur vous demande le Scrolling et appuyez ensuite simultanément sur les touches CAPS SHIFT et 1: de cette façon la ligne qui vous intéresse sera reportée sur la partie basse de l'écran et vous pourrez faire les modifications que vous désirez. Pour vous déplacer à droite ou à gauche, utilisez les touches 5 et 3 en même temps que CAPS SHIFT, tandis que, pour effacer le caractère à gauche du curseur utilisez CAPS SHIFT et Ø: pressez ENTER pour reintroduire la ligne à sa place. Afin d'obtenir plus de renseignements sur les fonctions "d'editing" nous conseillons de consulter le manuel Sinclair.
A présent, nous allons décrire en détail toutes les fonctions que le BASIC du Spectrum met

à notre disposition.
La première fonction que nous rencontrons est ABS, laquelle fournit la valeur absolue du nombre ou variable numérique utilisée comme argument, et qui entre autre prive le nombre du signe éventuel qui le précède, donc l'instruction:

PRINT ABS —91.4

produirait l'édition de 91.4. Cette fonction particulière peut se montrer très utile si on désire savoir si une variable contient un nombre positif ou négatif, il suffit en effet de procéder de la manière suivante:

IF A = ABS A THEN...

De cette façon les instructions qui suivent THEN ne seront exécutées que si A contient une valeur positive. Essayez d'en comprendre vous-mêmes la raison.
La fonction SGN au contraire détermine directement le signe d'une variable placée entre parenthèses et donne comme résultats:

 1 si positif
—1 si négatif
 0 si nul

Sachant cela nous pouvons reécrire la condition de l'exemple précédent de la façon suivante:

IF SGN A = 1 THEN...

En effet, le résultat de la fonction sera 1 si la variable contient un nombre positif.
Une autre fonction mathématique très employée est INT. Cette fonction donne la part entière du nombre donné. L'arrondissement est toujours fait à l'entier le plus bas, donc l'instruction:

PRINT INT 10.9

donnerait 10 comme résultat tandis que l'instruction:

PRINT INT —20.1

donnerait la valeur —21. Cette fonction particulière est souvent utilisée avec la fonction RND qui permet d'engendrer des nombres aléatoires ou pseudo-aléatoires. Le nombre engendré par cette fonction est toujours compris entre 0 et 1 (jamais 1). Mais avec un petit truc de programmation on peut procéder de façon à produire des séquences de nombres compris entre les limites qui nous intéressent le plus. En effet si nous multiplions le résultat de la fonction RND par le nombre maximum qui nous intéresse, nous obtiendrons justement les nombres compris entre 0 et le nombre spécifié.
Voyons un exemple:

A = RND * 20

De cette façon nous obtiendrons que la variable A contienne seulement des nombres compris entre 0 et 19. Mais si nous avions voulu que la séquence de nombres ne parte pas de 0 mais d'une autre valeur selon notre choix? Rien de plus facile! Il suffit en effet d'additionner le nombre le plus faible de la série au résultat de la fonction RND, multiplié par l'intervalle existant entre le minimum et le maximum.



Eclaircissons ceci par un simple exemple: si nous voulions engendrer uniquement des nombres aléatoires compris entre 50 et 80 nous procéderions de la manière suivante:
A = 50 + RND* + (80—50 + 1)
Cette formule est toujours valable et pourra se démontrer utile en différentes occasions. A la différence des deux limites remarquez que l'on doit ajouter 1 car autrement la fonction arriverait au maximum à 79. Faites cependant attention car de cette façon il sera engendré des nombres réels et donc si vous êtes seulement intéressé par les entiers, il faudra que vous fassiez précéder le tout par la fonction INT que nous venons à peine d'étudier, et écrire le tout comme suit:
A = INT (50 + RND * (31))
où 31 n'est autre que la différence entre 80 et 50 augmentée d'une unité.
En outre, le Spectrum dispose d'une fonction BIN très pratique, qui permet de convertir les nombres binaires en nombres décimaux. Si nous écrivons:
PRINT BIN 110
nous obtiendrons l'édition de 6.
Outre ces fonctions d'utilisation générale, l'ordinateur dispose d'une série complète de fonctions trigonométriques et mathématiques, qui sont: SIN pour le sinus, COS pour le cosinus, TAN pour la tangente, ATC pour l'arctangente, LOG pour les logarithmes, EXP pour l'exponentielle et SOR pour les racines carrées.
La majeure partie de ces fonctions n'ont besoin d'aucun commentaire, car elles se limitent à obtenir la valeur d'une fonction mathématique déterminée à partir de la valeur que nous fournissons. La seule qui mérite quelques précisions est la fonction EXP, laquelle rend la valeur de l'exponentielle de la constante 2.71828183, donc l'instruction:
PRINT EXP 7
équivaut à écrire
PRINT 2.71828183 ↑7
Toutefois toutes ces fonctions ne sont pas très fréquemment utilisées et peuvent intéresser uniquement celui qui veut utiliser son ordinateur pour des applications mathématiques.
Passons maintenant aux fonctions pour le traitement des chaînes, que le Spectrum peut facilement manipuler.
La fonction CODE donne comme résultat le code du premier caractère de la chaîne ou variable alphanumérique mise entre parenthèses.
Pour connaître les codes correspondants aux caractères il faut consulter un barème.
La fonction opposée à CODE est CHR$. Elle fournit le caractère correspondant au code numérique utilisé comme argument.
Une fonction très utile est LEN, en effet elle permet de connaître la longueur de la chaîne en examen. Par exemple, si dans un programme

nous définissons une variable alphanumérique, nous pouvons ensuite savoir de combien de caractères elle est composée en appliquant cette fonction. Donnons un exemple:

```
10 LET A$ = "ORDINATEUR"
20 PRINT LEN A$
```

De cette façon nous obtiendrons l'impression de 10 qui est justement la longueur de la chaîne A$.
Voyons maintenant le point fort de l'élaboration des chaînes: l'extraction et la manipulation des parties formant une chaîne principale.
La fonction qui nous permet de traiter complètement les chaînes est TO (il suffit en effet de spécifier le premier et le dernier caractère à extraire). Reportons-nous à la chaîne A$, précédemment définie et écrivons:
PRINT A$ (1 TO 4)
nous obtenons comme résultat la chaîne ORDI qui correspond justement aux caractères de 1 à 4 de la chaîne principale ORDINATEUR. Dans ce cas, puisque le premier caractère à extraire est le premier de la chaîne nous pouvons omettre le point de départ et écrire seulement A$ (TO 4)
Si, au contraire, le dernier caractère à extraire est aussi le dernier de la chaîne principale, nous pourrons en omettre le nombre et écrire
PRINT A$ (5 TO)
et nous produirons ainsi l'édition de NATEUR. On obtiendrait le même résultat en écrivant PRINT A$ (5 TO 10).
Si on désire extraire un seul caractère, il suffit de spécifier sa position à l'intérieur de la chaîne principale, si nous écrivons:
PRINT A$ (3)
nous verrons apparaître la lettre D qui, justement, représente le troisième caractère à l'intérieur de la chaîne A$.

Bien, nous avons terminé. Dans le prochain numéro, nous nous occuperons des instructions pour la gestion graphique ainsi que de plusieurs fonctions particulières.



LE LANGAGE MACHINE

Pour apprendre à connaître le langage machine nous devons au préalable souligner les concepts de l'électronique digitale, TTL, système binaire et système hexadécimal. De plus, il sera opportun d'approfondir la connaissance du microprocesseur. Ce sera le sujet de notre première leçon.

1.1. L'ELECTRONIQUE DIGITALE

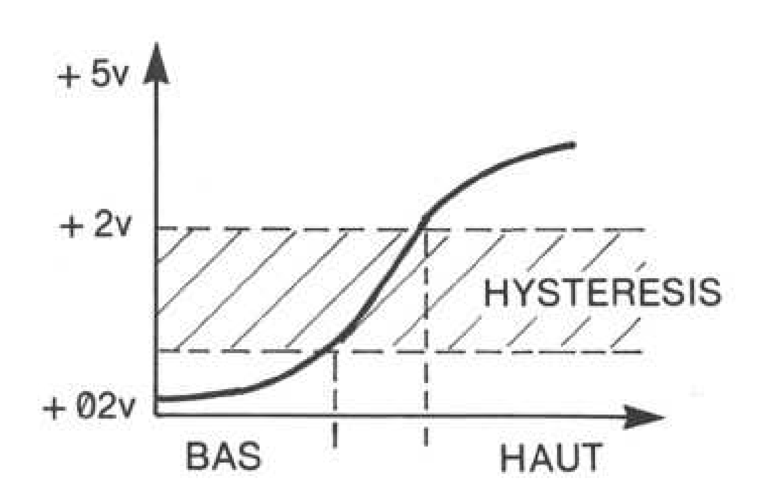
Avant de nous aventurer dans le "coeur" du microprocesseur (cerveau de l'ordinateur), je pense qu'il serait utile, pour qui aborde pour la première fois le langage machine, de donner quelques notions de base qui, par la suite, permettront un apprentissage plus simple et plus complet du langage machine proprement dit. Il existe dans l'électronique deux secteurs principaux: le digital et l'analogique.

Tandis que pour l'électronique analogique on considère les variations de différences de potentiel (les fameux volts), dans l'électronique digitale on a seulement deux différences de potentiel bien définies: c'est-à-dire: en électronique analogique un dispositif peut engendrer un champ continu de volts (par ex. de Ø à 2Ø volts), tandis qu'un dispositif digital ne peut engendrer que deux différences de potentiel bien définies, qui varient selon la famille auquel le dispositif appartient, mais qui à l'intèrieur de cette famille sont bien définis et standard. Pour vous donner un exemple, nous pouvons dire qu'un intégré digital (ces étranges boîtiers noirs en plastique avec deux rangées de pattes) qui se trouvent à l'intèrieur de tout ordinateur et qui contiennent un ou plusieurs dispositifs digitaux), de la famille des TTL, ne peut engendrer que deux valeurs de tension en sortie, Ø volts ou + 5 volts. Naturellement il existe plusieurs familles d'intégrés digitaux tels que DTL, ECL, TTL, RTL etc. L'appartenance d'un intégré à une certaine famille dépend uniquement de l'architecture interne et des standards de tension nécessaires au dispositif lui-même, mais évitons ce discours qui, dans un espace aussi restreint, serait trop vaste et trop complexe à expliquer.

1.2. LA FAMILLE DES TTL.

La famille d'intégrés qui nous intéressent plus particulièrement est celle des TTL (Transistor-Transistor-Logic) car c'est la plus utilisée dans le projet et la réalisation des ordinateurs. Nous

avons déjà indiqué qu'un dispositif TTL ne peut engendrer qu'un signal de 0 volts ou + 5 volts. Mais cela ne se produit qu'en théorie, car ces tensions à cause de différents facteurs internes et externes peuvent varier d'un léger pourcentage. Mais ce qui nous intéresse le plus est que le dispositif est capable d'engendrer deux tensions bien séparées entre elles par un intervalle appelé "hystérésis" du dispositif.
Sur le diagramme on remarque comment l'impulsion de tension qui, en théorie, devrait être de + 5 volts (niveau "haut") peut varier d'un minimum de + 2 volts à un maximum de +5 volts, tandis que l'impulsion de tension qui devrait être de 0 volts (niveau "bas") peut atteindre un maximum de +0,8 volts.
Si au niveau bas de tension nous attribuons le nombre 0 et au niveau haut le nombre 1, nous obtenons qu'un dispositif digital peut engendrer un nombre qui a la valeur de 0 ou de 1, appelé BIT. Ceci explique pourquoi tous les ordinateurs prennent pour base la numération binaire, ce qui permet des opérations sur des nombres formés de deux seuls symboles, 0 ou 1.



1.3. LE SYSTEME BINAIRE

Chaque système de numération est caractérisé par un nombre b, appelé base du système, qui équivaut au nombre de symboles disponibles dans le système donné pour représenter un nombre. Dans le système décimal, b équivaut à 10, puisque nous avons à disposition dix symboles (de 0 à 9), dont les combinaisons nous permettent de représenter n'importe quel nombre. Une règle



dont il faut se souvenir est que si on a "n" chiffres qui forment un nombre, les combinaisons possibles avec ces "n" chiffres sont égales à la base b élevée à n (b^n). En effet, en base décimale, avec quatre chiffres nous pouvons représenter $10^4 = 10000$ combinaisons différentes de nombres (de 0000 à 9999). Par ailleurs l'une de ces combinaisons, par exemple 3658, est décomposable de la manière suivante:

$$3658 = 3 \times 1000 + 6 \times 100 + 5 \times 10 + 8$$

c'est-à-dire:

$$3658 = 3 \times 10^3 + 6 \times 10^2 + 5 \times 10^1 + 8 \times 10^0$$

En reprenant le discours précédent nous remarquons que $10^2 = 10^{h-1}$, $10^2 = 10^{h-2}$, et ainsi de suite; donc:

$$3658 = 3 \times 10^{h-1} + 6 \times 10^{h-2} + 5 \times 10^{h-3} + 8 \times 10^{h-4}$$

Considérons maintenant les chiffres 3,6,8,5 en tant que coefficients, et appelons-les a. Il s'ensuit que n'importe quel nombre A peut être représenté sous la forme suivante:

$$A = a_{h-1} \times b^{h-1} + a_{h-2} \times b^{h-4} + ...a_1 \times b^1 \times a_0 \times b^0$$

Cette dernière formule (du polynôme) est applicable à n'importe quel système de numération, donc, un nombre binaire de 4 chiffres (1011) est ainsi décomposable:

$$1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Ce calcul donne comme résultat 11 en base décimale (11_{10}). Ainsi nous pouvons, sans problème, convertir un chiffre du binaire au décimal et, par conséquent, un dispositif digital peut facilement représenter un nombre décimal. Si, par exemple, nous avons trois lignes électriques différentes, et dans chacune d'elles, nous avons soit 0 volts soit +5 volts nous avons une combinaison binaire. En effet, si 0 volts = 0 et +5 volts = 1, il s'ensuit que:

+ 5V __________ 1
+ 0V __________ 0
+ 0V __________ 1

Nous en concluons que, trois lignes électriques peuvent représenter le nombre binaire 101_2 (101_2 pour dire 101 en base 2), qui équivaut au nombre décimal 5_{10}(5_{10} pour dire 5 en base 10). Dans ce cas, comme dans le système décimal, après avoir donné n chiffres, on peut calculer le nombre de combinaisons possibles en élevant la base b à n,b. Et donc, dans un système binaire avec trois chiffres, on peut obtenir un maximum de $2^3 = 8$ combinaisons, avec quatre chiffres un maximum de $2^4 = 16$ combinaisons.

1.4. LE SYSTEME HEXADECIMAL.

Après avoir appris à transformer un nombre binaire en un nombre hexadécimal, nous nous trouvons en face d'un autre problème; en effet, un programmeur qui s'aventure pour la première fois dans le langage machine remarque qu'un nombre décimal peut être représenté au moyen de quatre bits, mais les combinaisons de ces unités binaires sont $2^4 = 16$, et donc, bien supérieures aux dix combinaisons que demande le système décimal. En effet, en représentant toutes les combinaisons de quatre bits et en mettant à côté la conversion décimale correspondante, nous obtenons:

binaire	décimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	?
1011	?
1100	?
1101	?
1110	?
1111	?

(figure 1.4/1)

On remarque donc qu'il n'existe aucun symbole pour représenter les combinaisons supérieures à neuf (certains penseront à 10, 11, 12 ..., mais je vous rappelle que ceux-ci sont des nombres et non pas des symboles).

C'est pourquoi, afin de former avec les 10 symboles décimaux le système hexadécimal (base 16), il est nécessaire d'introduire d'autres symboles qui sont les six premières lettres de l'alphabet (A-F), ce qui nous donne le tableau suivant:

binaire (base 2)	décimal (base 10)	hexadécimal (base 16)
0000	0	0
0001	1	1
0010	2	2
0011	3	3



0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	'10'	A
1011	'11'	B
1100	'12'	C
1101	'13'	D
1110	'14'	E
1111	'15'	F

(figure 1.4/2)

Si quelqu'un pense encore que les nombres 10,11...etc peuvent résoudre ce problème tout aussi bien que l'hexadécimal, nous lui ferons remarquer que si nous avons un nombre binaire à huit bits, nous pouvons le subdiviser en deux nombres de 4 bits chacun (nibble).
Par exemple:

10110010 peut être divisé en 1011 et 0010.

En représentant ces deux nibbles avec deux nombres décimaux, il résulterait ceci:

$1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11_{10}$
$0010 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2_{10}$

En accouplant ensuite les deux nombres obtenus on obtient comme résultat 112.
Si nous calculons maintenant la véritable valeur du nombre à 8 bits nous obtenons.

$10110010 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^0 = 178_{10}$

Nous notons donc, qu'en décimal le nombre obtenu des nibbles (112) ne correspond pas à la valeur réelle de l'octet. Si nous employons le système hexadécimal sur les nibbles nous obtenons le nombre B2H(H = hexadécimal).
En obtenant maintenant avec la formule du polynôme la valeur décimale correspondante de B2 nous avons:

$B2_{16} = B_{10} \times 16^1 + 2_{16} \times 16^0$

En rappelant que B_{16} correspond à la valeur 11_{10} (fig. 1.4/2) il s'ensuit:

$B2_{16} = 11 \times 16^1 + 2 \times 16^0 = 178_{10}$

ce qui correspond à la valeur réelle de l'octet.
Donc, avec le système hexadécimal nous pouvons représenter directement des nombres binaires formés de multiples de 4 bits au moyen de symboles, tandis qu'avec le système décimal nous devrons obtenir la valeur du nombre au moyen d'une formule arithmétique.

1.5 LES MICROPROCESSEURS

Ce n'est pas sans raison si jusqu'à présent nous vous avons parlé d'octets, de bits et de TTL. En effet, tout cela sert à vous initier au UP (Microprocesseur), niveau supèrieur à celui atteint par ceux qui ont quelque expérience dans l'informatique.
Notre but est de vous faire comprendre, outre le langage machine, ses connexions avec la partie "hard" (la partie technique proprement dite). Un UP doit exercer toutes les fonctions mathématiques et de transmission de données à l'intèrieur d'un ordinateur. Externement le UP se présente comme un TTL normal, de dimensions importantes, dont le boîter (DIP), est soit en plastique, et souvent en céramique. A l'intèrieur se trouve le "chip", une plaque de silicium, de dimensions réduites (quelques mm. carrés). Ce chip est connecté électriquement aux pattes qui sortent des deux côtés du DIP. Ces pattes relient électriquement le circuit intégré au circuit imprimé (la plaque avec des pistes en cuivre sur lesquelles sont montés les composants) de façon à créer différentes connections entre tous les circuits intégrés.
L'ensemble de ces connexions est appelé "chaîne logique" et permet la transmission des données et la synchronisation entre les différents circuits intégrés et dispositifs externes (périphériques).
Chaque patte d'un Up reçoit et transmet des données ou instructions depuis et pour la CPU (une autre manière pour définir le microprocesseur, dérivé de Central Processing Unit). Cet ensemble de pattes est appelé "donnée Bus" et le fonctionnement est assez simple.
Supposons que ce "Bus" soit formé de 8 pattes (dans ce cas le UP s'appelle à 8 bits) et que nous fournissions à ces pattes une combinaison de niveaux hauts et bas (Ø volts et + 5 volts). Chaque combinaison que nous fournissons sur le Bus peut correspondre à une donnée numérique ou à des instructions que la CPU doit élaborer. En expliquant ceci au moyen d'un schéma, nous obtenons la figure 1.5/1, où nous trouvons 8 lignes électriques (la donnée Bus) qui se connectent à l'Up.



+ 5v	1	
+ 0v	0	
+ 0v	0	
+ 5v	1	
+ 0v	0	MICROPROCESSEUR
+ 5v	1	
+ 5v	1	
+ 5v	1	

(figure 1.5/1)

La combinaison des signaux donne comme résultat un nombre binaire, (10010111 fig. 1.5/1) qui a pour la Up un sens bien précis et qui peut être, soit une donnée numérique sur laquelle travailler, soit une instruction à exécuter. Nous en déduisons que:
1) L'Up (par conséquent le langage machine) ne fait aucune distinction entre les données numériques sur lesquelles travailler et sur les instructions à exécuter. En effet toutes les deux sont des combinaisons binaires.
2) Le langage machine se réduit toujours à un ensemble de nombres. C'est le programmeur qui doit les placer de façon à ce que la Up exécute le programme correctement (nous verrons comment par la suite). Pour démontrer comment les nombres peuvent représenter des instructions prenons l'exemple du nombre 76_{10} (118_{10}). Ce nombre envoyé sur le Bus de données d'une CPU Z-80 (Z-80 est le nom), arrête toute élaboration de cette dernière qui reste dans l'attente qu'une condition advienne, ou que la donnée $F8_{16}$ (243_{10}), toujours sur Z-80, empêche que la CPU soit interrompue au cours d'une élaboration grâce à un dispositif externe.
Naturellement la correspondance entre codes numériques et instructions n'est pas universelle pour tous les Up, mais varie selon le type de la CPU. En outre les données à envoyer sur une donnée Bus d'un Up varient selon les dimensions du Bus lui-même, d'où la dénomination d'un Up à 8, 16, 32 bits, selon justement qu'il accepte les combinaisons de 8,16,32 lignes électriques en entrée comme données.
Voici les noms de plusieurs microprocesseurs parmi les plus utilisés:

6510	8 bits
Z-80	8 bits
8080	8 bits
8085	8 bits
68000	16 bits
TMS 9900	16 bits
Z 8000	16 bits

(figure 1.5/2)

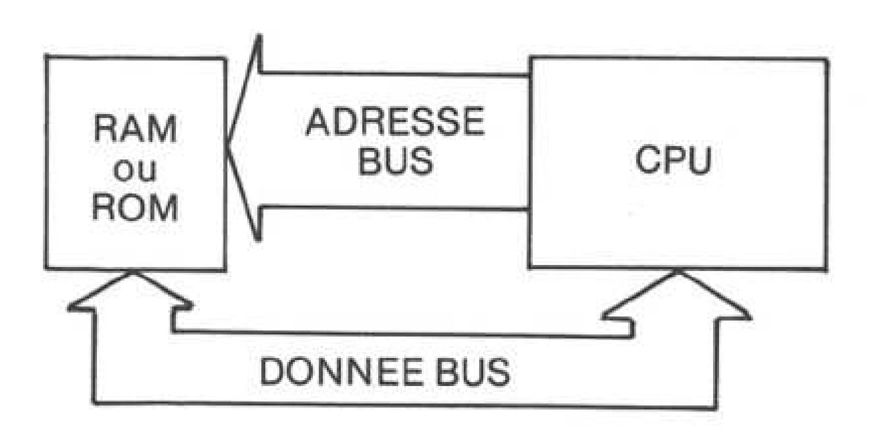
A présent, après avoir appris comment on envoie une instruction à un Up, il faut expliquer comment ce dernier peut trouver la série d'instructions à exécuter. Outre la "donnée Bus", il existe une "adresse Bus" dont la constitution est semblable à la "donnée Bus", mais où la direction des données est seulement dirigée vers les mémoires. Les combinaisons de Bits sur ce Bus, au lieu de représenter des instructions, vont "piloter" des mémoires externes. Elles sont formées de petites cases qui ont la capacité de mémoriser un octet et qui sont numérotées. Les différentes combinaisons sur "l'adresse Bus" servent à sélectionner l'une de ces cases. Après avoir sélectionné une case, son contenu est placé de la mémoire sur le Bus de données, permettant aussi à la CPU de le lire. Naturellement la CPU, après avoir sélectionné une case, peut aussi y écrire à l'intèrieur et la mémoire devra recueillir la donnée à mémoriser à partir de la donnée Bus. Nous remarquons donc une différence entre Donnée Bus et Adresse Bus. En effet, dans la premiere les données peuvent être transmises depuis la mémoire jusqu'à la CPU et vice-versa. C'est pourquoi ce Bus est appelé "bidimensionnel" (vers deux directions de flux de données), tandis que l'Adresse Bus est "unidirectionnel", car le flux d'adresses va uniquement vers la mémoire. Nous rappelons que l'Adresse Bus peut varier en grandeur (de même pour la Donnée Bus) selon la UP à laquelle elle appartient. En général, un UP à 8 bits possède une Adresse Bus de 16 bits, dont la combinaison peut adresser $2^{16} = 65536$ cases de mémoire. Donc, plus une Adresse Bus est grande, plus grande est la capacité de mémoire accessible depuis une CPU. Voici à présent les dimensions des Adresses Bus des microprocesseurs précédents:

uP	Donnée Bus	Adr. Bus
6510	8 bits	16 bits
Z-80	8 bits	16 bits
8080	8 bits	16 bits
8085	8 bits	16 bits
68000	16 bits	24 bits
TMS 9900	16 bits	16 bits
Z-8000	16 bits	16/23 bits

Voici en outre un schéma flux données des connexions entre mémoire externe et CPU.



La flèche de l'Adresse Bus indique un flux unique de données (les adresses) vers la mémoire, tandis que la double flèche de la Donnée Bus indique les deux directions possibles du flux des données.
Ceci dit, terminons cette première partie sur le l/m, en espérant avoir été suffisemment clair avec cette introduction. Elle vous sera utile pour les chapitres suivants sur le cours l/m et sur tous les Up en général.
à suivre...



INSTRUCTIONS POUR LA CASSETTE DE SOFTHEQUE n° 5

INSTRUCTIONS DE CHARGEMENT

Côté ZX SPECTRUM

Pour charger les programmes introduisez la cassette dans le magnétophone et écrivez "LOAD" (s'obtiennent avec SYMBOL SHIFT et P) immédiatement après appuyez sur la touche ENTER et faites marcher le magnétophone avec PLAY.

Ensuite il suffira de suivre les instructions qui apparaitront sur vidéo. Lorsque vous aurez fini d'utiliser un programme éteignez l'ordinateur en détachant pendant un instant le câble d'alimentation. Tapez encore une fois sur LOAD suivi de ENTER afin de charger le programme suivant.

3D GRAPH

3D GRAPH est un programme scientifique pour dessiner sur trois dimensions z=(x,y) n'importe quelle fonction. Le résultat sera en général une surface.

Le programme demande:

1) définir la fonction
2) définir les limites
3) angle d'élévation
4) angle de rotation
5) définition des lignes
6) résolution sur les axes
7) blank lignes

Voyons un exemple:

1) z=f(x,y)=x*x-y*y
2) xmin=-10 xmax=10
   ymin=-10 ymax=10
   A présent le programme vous demandera d'attendre.
   Après quoi, si tout va bien on repartira.
3) Cet angle concerne l'observateur et sert à voir la figure sous différents points de vue.



4) Comme ci-dessus.
Rotation = 18
5) Admettons (pour la définition de la grille):
lignes x = 12
lignes y = 12
(on conseille de toujours donner des valeurs symétriques)
6) Plus la résolution est haute et plus le programme est lent; admettons:
résolution x = 13
résolution y = 13
7) En répondant "y" nous obtiendrons la suppression des lignes cachées.
Si ce n'est pas absolument nécessaire, nous déconseillons ce choix.
En effet, en disposant "y" le programme deviendra extrèmement lent.
Le graphique terminé, en appuyant sur C nous obtiendrons une copie sur l'imprimante.
Les exemples de fonctions sont:
1) x*x/2 + y*y/3 + 2
2) (x*x-y*y)*(x*x-y*y)
3) x*y + sin(x*y)
Et bien d'autres encore...

TRAFIC SPATIAL

Encore une fois vous vous trouvez, perdu avec vos astronefs super-rapides, dans un espace fantastique et dangereux, peuplé d'imprenables fantômes et de monstres furieux qui essaient de vous supprimer sans pitié. Pour commencer la "lutte", appuyez sur la touche 1. Pour éviter les ennemis, vous pourrez vous déplacer avec agilité en utilisant la touche 9 pour avancer; la touche 7 pour descendre. Pour faire feu avec votre laser, appuyez sur la touche 0. Rappelez-vous, vous ne disposez que de quatre astronefs pour lutter contre les fantômes qui arrivent de toutes parts et contre les monstres qui vous poursuivent de plus en plus nombreux et toujours plus rapides! A la fin du conflit, dans lequel vous devrez, naturellement, essayer d'obtenir le score maximum, vous pourrez mettre votre nom dans la liste des meilleurs "guerriers de l'Espace" (en centrant les lettres avec les touches 6 et 7 et en tapant sur la touche 0 pour obtenir l'inscription).

Rappelez-vous que dans ce jeu, vous pourrez défier vos amis et démontrer qui est le guerrier plus habile de l'Espace. A la fin

de la partie, vous pourrez contrôler votre score ou le confronter à celui de vos compagnons de jeu, en occupant les douze espaces permis pour écrire le nom. Et puis, en route pour une nouvelle aventure dans l'espace!

INSTRUCTIONS DE CHARGEMENT

Côté VIC 20

Prendre la cassette, la mettre dans le magnétophone avec la face choisie, A ou B, vers le haut.

Allumer l'ordinateur, écrire l'instruction 'LOAD' et appuyer sur la touche 'return': l'inscription "PRESS PLAY ON TAPE" apparaîtra; elle signifie "appuyer sur la touche PLAY sur le magnétophone". Exécuter l'ordre et attendre que l'inscription "FOUND INTRODUCTION A" apparaisse sur l'écran; à ce point, appuyer sur la touche d'espacement et attendre quelques secondes, jusqu'à ce que l'écran réapparaisse avec le curseur clignotant.

Si une inscription d'erreur apparaît, ré-enrouler la cassette et recommencer les opérations.

Si tout est O.K., appuyer sur la touche "STOP" du magnétophone, écrire sur l'écran "RUN" et taper "RETURN" sur l'ordinateur; après quelques instants apparaît la première page de notre revue avec la liste des programmes contenus dans la cassette. Lorsque apparaît l'inscription "ENFONCER LA TOUCHE", appuyer sur une touche du clavier au hasard. Dès que l'écran devient bleu, avec le bord bleu-clair et les inscriptions dans la partie supérieure que vous voyez normalement en allumant l'ordinateur , écrivez de nouveau "LOAD" et appuyez sur "RETURN"; appuyez sur "PLAY" sur le magnétophone et attendez que l'ordinateur trouve le programme suivant, appuyez sur la touche d'espacement et attendez un instant jusqu'à ce que le curseur clignotant réapparaisse. Si aucune inscription d'erreur n'apparaît, appuyez sur "STOP" sur le magnétophone, formez "RUN" sur l'écran et appuyez sur "RETURN" sur l'ordinateur, afin de faire démarrer le programme. Suivez ces simples instructions chaque fois que vous voulez charger un programme du softhèque ordinateur.

Si vous aviez du mal à charger les programmes et si des inscriptions d'erreur (ex. PRINT LOAD ERROR) apparaissaient sur l'écran, vous pourriez essayer de modifier la position des deux têtes à l'aide d'un tournevis introduit dans le trou situé dans la partie supérieure du magnétophone.



Après quelques essais, vous ne devriez plus avoir de problèmes. Si vous voulez changer de programme, vous avez une alternative: A) appuyez sur la touche: RUN/STOP; si rien ne se produit, en la maintenant toujours enfoncée, tapez une ou plusieurs fois sur la touche "RESTORE": l'écran devrait se vider et redevenir bleu avec le bord bleu clair. Alors écrivez "LOAD" et suivez les instructions habituelles. B) si vous n'arrivez vraiment pas à vous en sortir, n'ayez crainte, éteignez l'ordinateur et réallumez-le, et recommencez en suivant les instructions habituelles pour charger les programmes.

LES PONTS

Après avoir introduit notre jeu, lisez attentivement les instructions; vous pourrez utiliser le joystick ou encore les lettres E R T D G C V B avec le schéma reproduit sur votre écran et que vous aurez intérêt à transcrire pour mieux vous le rappeler. Ensuite choisissez le niveau de difficulté (1/5). Au début ne soyez pas trop sûr de vous, car le jeu n'est pas aussi simple qu'il en a l'air.

Vous vous rendrez bien vite compte que ce petit homme malhabile ne va jamais du côté où vous vous attendez qu'il aille, et si vous n'êtes pas suffisamment habile et rapide, vous n'empêcherez pas votre protégé de se précipiter dans les eaux froides du fleuve qui coulent sous les ponts que vous devrez construire, comme si vous aviez dix mains. Attention car les ponts ne restent pas debouts très longtemps et vous n'aurez pas un seul instant pour respirer. Courage! Un habile ingénieur comme vous ne peut certainement pas se laisser impressionner par les difficultés!

PUZZLE

Voici un passe-temps amusant et utile pour entraîner votre mémoire visuelle.

L'image de départ est un clair paysage de printemps: ciel bleu, soleil, un arbre chargé de fleurs et une route de campagne qui se perd au lointain. Par la suite, on vous montrera le même paysage subdivisé en neuf morceaux placés au hasard et même quelquefois à l'envers: Vous devrez les replacer afin de reconstituer l'image du début.

Pour changer la position de deux morceaux, appuyez successivement sur les deux numéros qui les indiquent; pour redresser un morceau placé de travers, appuyez sur son numéro et sur n'importe quelle autre touche.

EASY WORD

EASY WORD est un programme qui vous permet de découvrir de nouvelles qualités de votre ordinateur. Le programme transforme le VIC 20 en une machine à écrire électronique qui peut effectuer les corrections directement sur vidéo au moyen de la touche DEL. En tenant enfoncée la touche SHIFT, on obtiendra les lettres majuscules que l'on pourra bloquer avec la combinaison SHIFT LOCK. Pour aller à la ligne, il suffit d'actionner la touche RETURN. Avec HOME, on retourne à la première ligne et avec CLR, on efface la touche en mémoire.

En utilisant les touches CRSR UP et CRSR DOWN, on déplace l'image de la feuille dans la machine et avec CTRL + 1, CTRL + 2 et CTRL + 3, on change l'espace entre les lignes. L'espace choisi apparaît avec le numéro à gauche du chariot.

Une fois le texte tapé, celui-ci peut être sauvergardé sur bande, sur disque ou encore directement imprimé avec l'imprimante.

ATTENTION: après avoir tapé 40 lignes, la mémoire commence à manquer; il vaudra mieux sauvergarder le texte afin d'éviter...un travail inutile.

SEVEN ELEVEN

C'est le jeu de hasard préféré des assidus de tous les grands Casinos. Cette fois-ci, vous pourrez défier votre ordinateur dans une passionnante partie à deux.

Vous commencez la partie avec un capital initial de 100$ chacun: à chaque coup, 10$ sont mis en jeu et seront ajoutés ou supprimés selon les victoires ou les défaites.

Qui tient la banque devra lancer les dés le premier: avec les combinaisons obtenues, le 12, le 2 et le 3 perdent immédiatement, le 7 et le 11 gagnent. Si un des nombres restants sort, il devra relancer les dés jusqu'à obtenir pour la seconde fois le nombre sorti précédemment: cependant, si au cours des coups suivants le 7 sort, il perd.

En cas de défaite, la banque passe aux mains de l'adversaire. Par contre, si vous gagnez, vous pourrez choisir si continuer à tenir la banque (appuyez alors sur la touche S à la demande de l'ordinateur) ou passer (touche N).

Pour commencer à jouer appuyez sur une touche:

Bonne chance et amusez-vous bien!



Numéros déjà parus:

SOFTHEQUE N. 1
Pour les possesseurs de ZX SPECTRUM et VIC 20
Dans la revue n° 1: Le pseudocode et la programmation de base - Les touches fonctionnelles et le joystick du Vic 20 - Comment augmenter la vitesse des programmes en Basic - La mémorisation des données du ZX Spectrum - Sinclair et Commodore: les nouveautés.
Dans la cassette n° 1: Côté ZX Spectrum: U.F.O - Wargame - Anatomie - Gestion Magasin.
Côté Vic 20: Quotient Intellectuel - Chasse au Trésor - Enfer 3D - Gestion Magasin.

LOGITHEQUE N. 1
Pour les possesseurs de Commodore 64 et TI 99/4A
Dans la revue n° 1: Le pseudocode et la programmation de base - Les touches fonctionnelles du Commodore 64 - Le Basic c'est quoi au juste? - TI pour le son.
Dans la cassette n° 1: Côté Commodore 64: Wargame - Slalom - Quotient Intellectuel - Budget Familial.
Côté TI 99/4A: Splat - Poker - Puissance 4 - Voyage dans l'Espace.

SOFTHEQUE N. 2
Dans la revue n° 2: Pseudocode: Théorie et application des matrices - Qu'est-ce qu'un langage de programmation - Comment développer un programme.
Dans la cassette n° 2: Côté ZX Spectrum: Tour Laser - Course de Grenouilles - Hélibomber - Régime et calories.
Côté Vic 20: Régime - Calories - Tir à la Cible - Go Moku.

LOGITHEQUE N. 2
Dans la revue n° 2: Pseudocode: Théorie et application des matrices - Qu'est-ce qu'un langage de programmation? - Comment développer un programme?
Dans la cassette n° 2: Côté Commodore 64: Formule 1 - Black Jack - Tennis 3D - Gestion Magasin.
Côté TI 99/4A: Labyrinthe - Slot Machine - Agent Secret - Space War.

SOFTHEQUE N. 3
Dans la revue n° 3: Pseudocode: Instructions read et data - Les secrets du Vic 20 - Cours de Basic.
Dans la cassette: Côté ZX Spectrum: Mixxil - Sequencer - Biorythme - Ping Pong.
Côté Vic 20: Attaque Aérienne - Stop Music - Memory Trainer - Change.

LOGITHEQUE N. 3
Dans la revue n° 3: Pseudocode: Instructions read et data - Périphériques d'entrée et sortie - Cours de Basic - Améliorons nos programmes sur le TI 99/4A.
Dans la cassette: Côté Commodore 64: Starway - Poker aux Dés - Régime - Calories.
Côté TI 99/4A: La Goulue - Comptabilité - Les Serpents - Le Coffre - Fort.

SOFTHEQUE N. 4
Dans la revue n° 4: Pseudocode: Structures données complexes - Queue - Liste - Simulation. Les systèmes de numération - Le Basic du Spectrum - Variables internes et Gestion Vidéo.
Dans la cassette: Côté ZX Spectrum: Bilan Familial - Les Gloutons - Ville Souterraine - Le Cerveau.
Côté Vic 20: Vic Calc - Quinze 3D - Abeilles - Horoscope.

LOGITHEQUE N. 4
Dans la revue n° 4: Pseudocode: Structures données complexes - Queue - Liste - Simulation. Les systèmes de numération - Les secrets du CBM 64 - Ti 99 4/A: Comment améliorer l'output sur l'écran.
Dans la cassette: Côté Commodore 64: Sink - Bois Enchanté - Mister Boa - Perspective.
Côté TI 99/4A: Lettres Assassines - Au Secours - Moyenne Scolaire - La Carte plus Forte.

Vous pouvez vous procurer les numéros déjà parus en envoyant un chèque bancaire ou postal à l'ordre de Promopublications (34 Champs Elysées 75008 Paris) d'un montant de 85 F + 10,70 F pour frais de port soit 95,70 F par numéro demandé.


SOFTHEQUE
ORDINATEUR
5
Vidéo-jeux et
programmes pour
ZX SPECTRUM
et VIC 20
ZX SPECTRUM
• 3D GRAPH
• TRAFIC SPATIAL
• LES MONSTRES
• AGENDA TELEPHONIQUE
VIC 20
• LES PONTS
• PUZZLE
• EASY WORD
• SEVEN ELEVEN